Name: Amal Roy

Student ID: 11703660 (08)

**Problem:**

Q: Sudesh Sharma is a Linux expert who wants to have an online system where he can handle student queries. Since there can be multiple requests at any time he wishes to dedicate a fixed amount of time to every request so that everyone gets a fair share of his time. He will log into the system from 10am to 12am only. He wants to have separate requests queues for students and faculty. Implement a strategy for the same. The summary at the end of the session should include the total time he spent on handling queries and average query time

**Solution:**

The Problem Can be solved using two Algorithm:

* Using the First come First serve by giving the priority to the one who arrives first in the queue
* Using the round robin method by giving a particular time period to each student and the faculty

**Algorithm:**

* Start the Process
* Take the Total Number of queries
* Enter the time quantum of each process
* Enter whether the query is of student or faculty
* Enter the Details of the process for n queries
* Enter the arrival time of the process
* Check whether Arrival time > 1000 && Arrival Time < 1200
* If no Print (“Enter the valid arrival time”)
* If yes Enter the time required to resolve the query
* Resolving query:
* Enter resolving time
* If resolving time <= quantum time: process is completed
* If resolving time > Quantum time: resolving time = resolving time – quantum time: and add it at rear of queue
* Calculate the total time to resolve n queries
* Print the Total time taken to solve all the queries

**Code:**

#include<stdio.h>

struct job{

int pid;

int at;

int bt;

int cmpt;

int rbt;

}f[100], s[100], m[100];

int n, fc=0, sc=0, mc=0;

int quanta;

void roundRobin(){

int time= m[0].at, mark=0, cc=0, i, rc;

while(time!=120 && cc!=mc){

for(i=0; i<=mark; i++){

if(m[i].rbt > quanta){

time += quanta;

m[i].rbt -= quanta;

}

else if(m[i].rbt <=quanta && m[i].rbt !=0){

time += m[i].rbt;

m[i].rbt =0;

m[i].cmpt = time;

cc++;

}

else;

}

int start = mark+1;

for(rc= start; rc<mc; rc++){

if(m[rc].at <= time){

mark++;

}

}

}

}

void merger(){

int isc=0, ifc= 0, min, flag;

if( fc!=0 && sc!=0){

while(isc<sc && ifc<fc){

if(f[ifc].at == s[isc].at){

m[mc] = f[ifc];

mc++;

ifc++;

m[mc]= s[isc];

mc++;

isc++;

}

else if(f[ifc].at < s[isc].at){

m[mc]= f[ifc];

mc++;

ifc++;

}

else if(f[ifc].at > s[isc].at){

m[mc]= s[isc];

mc++;

isc++;

}

else;

}

if(mc != (fc+sc)){

if(fc!=ifc){

while(ifc!=fc){

m[mc]= f[ifc];

mc++;

ifc++;

}

}

else if(sc!=isc){

while(isc!=sc){

m[mc]= s[isc];

mc++;

isc++;

}

}

}

}

else if(fc==0){

while(isc!=sc){

m[mc]= s[isc];

mc++;

isc++;

}

}

else if(sc==0){

while(ifc!=fc){

m[mc]= f[ifc];

mc++;

ifc++;

}

}

else {

printf("\n No valid Jobs available\n");

}

}

void printer(){

int i=0, total=0, sum=0;

double avg;

printf("\nSummary for the Execution\n");

printf("\nQuery ID\tArrival Time\tRessolving Time\tCompletion Time\tTurn Around Time\tWaiting Time");

for(i; i<mc; i++){

printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t\t%d",

m[i].pid, (m[i].at+1000), m[i].bt, (m[i].cmpt+1000), (m[i].cmpt-m[i].at), ((m[i].cmpt-m[i].at)- m[i].bt));

total= m[i].cmpt;

sum+= (m[i].cmpt-m[i].at);

}

avg = sum/mc;

printf("\n\nTotal time Spent for all queries: %d", total);

printf("\nAverage query time: %lf", avg);

printf("\nProcess Execution Complete");

}

void input(){

int map, i, t;

printf("Enter total no of queries: "); scanf("%d", &n);

if(n==0) { printf("\n No queries\n"); }

else{

printf("\nEnter Quanta for each Process: "); scanf("%d", &quanta);

printf("\nEnter 1 for faculty and 2 for student\n");

for(i=0; i<n; i++){

printf("\nJob Type (1/2): "); scanf("%d", &map);

if(map==1){

printf("Query Id: "); scanf("%d", &f[fc].pid);

printf("Arrival Time: "); scanf("%d", &t);

if(t<1000 || t>1200){

printf("\nEnter Correct time");

input();

}

else{f[fc].at= t-1000;}

printf("Resolving Time: "); scanf("%d", &f[fc].bt); f[fc].rbt= f[fc].bt;

fc++;

} else{

printf("Query Id: "); scanf("%d", &s[sc].pid);

printf("Arrival Time: "); scanf("%d", &t);

if(t<1000 || t>1200){

printf("\nEnter Correct time\n");

input();

}

else {s[sc].at= t-1000; }

printf("Resolving Time: "); scanf("%d", &s[sc].bt); s[sc].rbt= s[sc].bt;

sc++;

}

}

}

}

void inst(){

printf("\nWelcome, please follow these instruction for proper functioning of the program"

"\n\*\*>Enter time in 2400 hours format. example for 10:30 am enter 10030"

"\n\*\*>Enter Query arrival times in ascending order, i.e., in real time arrival manner\n"

"\nAll Time units are in minutes. \n\n"

);

}

int main(){

inst();

input();

merger();

roundRobin();

printer();

}

**Output:**





